INTERFACING A NANONIS CONTROLLER WITH A SCANNING TUNNELING MICROSCOPE

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Justin Robert Rodriguez

February 2016

ABSTRACT

Scanning Tunneling Microscopy is an important tool in the study of condensed matter physics and has been aided by advances in computers. We obtained four Nanonis scanning tunneling microscope controllers to upgrade our current and new microscopes. The Nanonis hardware and software was designed to work with most scanning tunneling and atomic force microscopes. It has the ability to record multiple analog signals, transform and output similar signals, control piezoelectric motors, and monitor tip movement with a real-time operating system to provide necessary tip movements. The upgrade will afford us with more detailed scanning tools without sacrificing speed, as well as bring the benefits of newer hardware and support. We designed and built a software interface to complement the Nanonis hardware by providing a set workflow with new safety checks and to expose the additional functionality with a useful interface that fits our system. The scalable nature of our software will let us add new kinds of spectroscopic mapping that were unavailable to us in our older control system. Our current upgrade allows for more segmented energy ranges for bias spectrography scans to allow us to target areas of interest for more relevant data in less time. We discuss the structure of the underlying software and the hardware functionality and limits.

BIOGRAPHICAL SKETCH

Justin Rodriguez, raised in Syracuse NY, graduated from Christian Brothers Academy High School in 2006. He graduated from Cornell University in 2010 with a Bachelor of Arts double major in Physics and Computer Science. While at Cornell he did research in the Hoffstaetter group. Following attendance at Cornell, he joined a small company, Muons Inc., that focuses on particle accelerator research. While at Muons Inc. he worked at SLAC and Jefferson national labs with a supercomputing electromagnetic code, ACE3P. Justin returned to Cornell in 2012 to do a Masters of Science degree in the Applied Physics Department. He is currently attending Pennsylvania State University to earn his PhD.

This thesis is dedicated to all of my family for all their support while I was at Cornell, and to my fellow Applied Physics MS students for all the fun times and late nights we spent working together.

ACKNOWLEDGEMENTS

I would like to thank my advisor Professor Séamus Davis for giving me the opportunity to join this group and his guidance in getting started, and my committee chair Professor Lena Kourkoutis for all her support and suggestions on this thesis. A special thank you to Dr. Mohammad Hamidian who spent many hours helping find bugs in my code and helping me understand the system. I would also like to thank Dr. Kazuhiro Fujita, Dr. Freek Massee, and the rest of the Davis group for answering all of my questions and making me feel welcome. I wish to express my gratitude to Aitziber Herrero and Daniel Uehli at SPECS Zurich for answering my many support emails, and to my sister, Morgan, for all the grammar edits in this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

BACKGROUND

1.1 Scanning Tunneling Microscopy

Solid matter makes up most of the material we interact with on a daily basis. One of the first instruments used in the study of matter was the optical microscope to observe matter at small scales. However, due to its wave-like nature, light diffracts when passing through small apertures of a microscope. Any visible light used to see atomic features has too long of a wave length to be used in a microscope without diffracting, and any shorter wavelength light is often too energetic to shine on a material without damaging it. Later microscopes used electrons instead of light but were still not able to resolve individual atoms. To overcome these limits Gerd Binnig and Heinrich Rohrer developed the first Scanning Tunneling Microscope (STM) in 1981 that measured electrons moving through a needle-like tip.[2] For very sharp tips, STMs allowed for atomic scale resolution.

Scanning Tunneling Microscopy relies on the quantum mechanical nature of electrons. A STM uses a small, sharp tip that is positioned over a solid sample surface. A bias voltage between the surface and the tip is introduced to allow electrons to tunnel between the two. This looks in principle, very similar to the ideal 1D square potential barrier taught in introductory quantum mechanics. By measuring the amount of tunneling current between surface and tip it is possible to produce images mapping out the surface of a material. The tip is generally only a few atoms wide near the surface of the sample and can achieve a horizontal resolution of a few angstroms in ideal conditions. Figure 1.1 shows

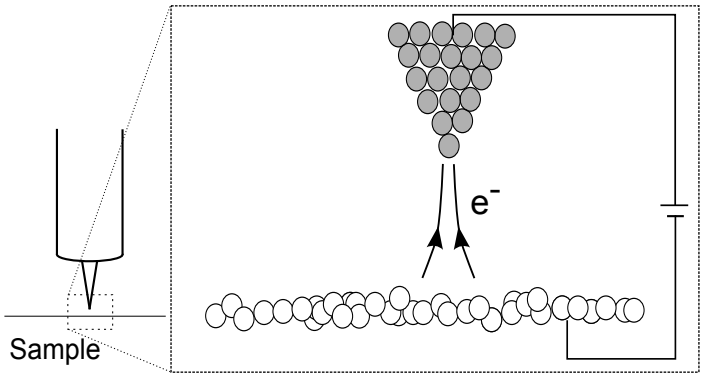


Figure 1.1: Model of a Scanning Tunneling Microscope
A simplified model of the sample and tip setup. The bias voltage between the sample and the tip cause electrons to tunnel between them.

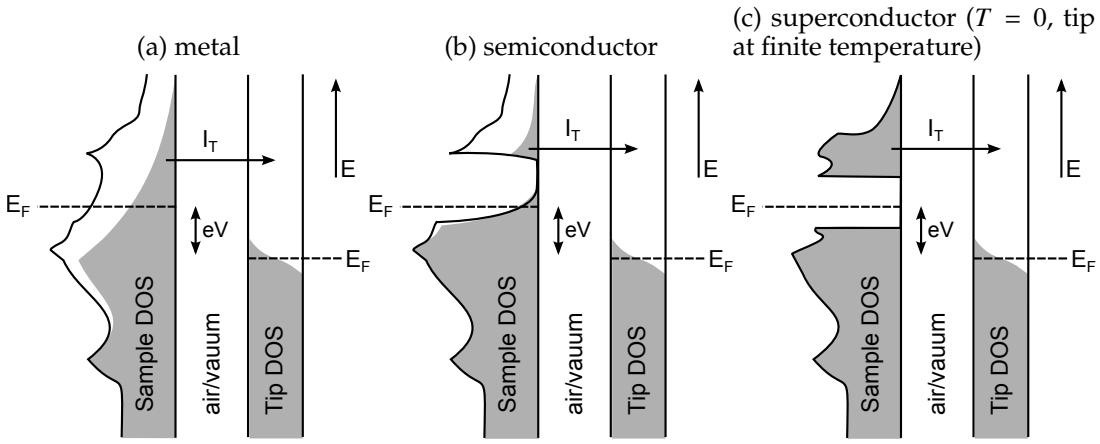
a simple model of a tip over a sample. For a roughly uniform material the tunneling current between the tip and the surface should be only a function of distance. To measure the topography of a roughly uniform sample the tip is moved towards the surface or away from it so that the current is kept constant at all times. By measuring the voltage needed to move the tip at each point we can produce an image of the sample topography.[12]

For a second measurement we can also lock the z position of the tip in place over a point of interest and slowly increase or decrease the bias while measuring tunneling current. We can then find the derivative of the current for the changing voltage either numerically or by using a lock-in amplifier. This is known as the dI/dV curve and is proportional to the local density of states (LDOS).[4] The dI/dV curve is given by

$$\frac{dI}{dV} \propto \rho\left(E_F - eV\right)$$

where E_F is the Fermi Energy, ρ is an arbitrary constant, and V is the voltage difference between the tip and sample.[12] The density of states gives us both the distribution of free electrons in the surface as well as a k-space mapping where we can see structures like the band structure. By measuring the interfer-

Figure 1.2: Density of states



The density of states between a tip and sample held at a voltage difference of eV. (a) shows a typical metal while (b) has the gap that we would expect to see in a semiconductor or insulator. (c) shows the gap at $T = 0K$ that is observed in superconductors. The density of states extends slightly above and below the Fermi energy E_F due to the voltage difference between the tip. In each case the Fermi energy is higher in the sample so electrons tunnel into the tip.

ence of the electron oscillations in the LDOS with waves scattered by defects we can map Fermi surfaces.[6] Figure 1.2 demonstrates a typical density of states interaction between a tip and a metal, semiconductor, and superconductor. For a semiconductor the density of states goes to zero for energies that fall within a gap. For a metal the density of states should always be finite for any bias voltage between the sample and tip. In a superconductor a gap forms when the material transitions below T_c and the density of states goes to zero at absolute zero.[4, 11, 8, 4]

1.2 Electronics

The Scanning Tunneling Microscope is managed through a separate controller computer that monitors signals, moves the tip, and adjusts the voltage and cur-

rent. The tip is moved using piezo electronic motors. The tip height can either be kept static or moved up or down in order to keep the STM at a constant tunneling current. This tip adjustment process is known as the feedback loop, and is one of the most important functions of the STM. By keeping the tip in feedback mode during the scan you can generate the topography of a sample. The controller is also responsible for guiding the tip towards the surface without crashing it into the surface, and thereby damaging the tip. The STM controller is also used in conjunction with a high voltage source and a lock-in amplifier for improving the signal to noise ratio and producing the dI/dV data.

1.2.1 Nanonis

The Davis group at Cornell and Brookhaven National Laboratory currently has three STMs and with a fourth under construction. The current microscopes are controlled by outdated hardware which is expensive to repair and subject to many of the limitations of the older software and hardware. The controller software is written in C for Windows 3.1 which lacked many of the programming libraries we use today. This greatly increases the time needed to make even the smallest changes. Windows 3.1 also has stability and performance issues that have been addressed in the 23 years since it was released. The microscope also uses PCI cards to connect to the computer that have been discontinued, making them difficult to replace. We purchased four Nanonis STM controllers from SPECS to replace the current hardware and add new functionality to the STM.

The Nanonis hardware is a modular, stand-alone controller running a real-time operating system running on a 3Ghz Pentium processor.[10] The controller

consists of several units with different functionality. Our setup includes the RC4 base station, SC4 analog input/output signal conversion, HVA4 high voltage amplifier, PMD4 piezo motor driver, and HVS4 high voltage supply. The base module connects to a windows computer via an Ethernet connection with TCP/IP/UDP. The base uses several dedicated digital connections to communicate with the signal converter, piezo motor driver, and high voltage amplifier.

The analog input/output signal conversion contains a National Instruments FPGA card with 8 input and output connections, each with 20 to 22-bit resolution.[9] The inputs and outputs range from ±10V and have a 100kHz bandwidth. SPECS also provides a Windows software interface to control the hardware, as well as an abstract programming interface (API) for custom programs.

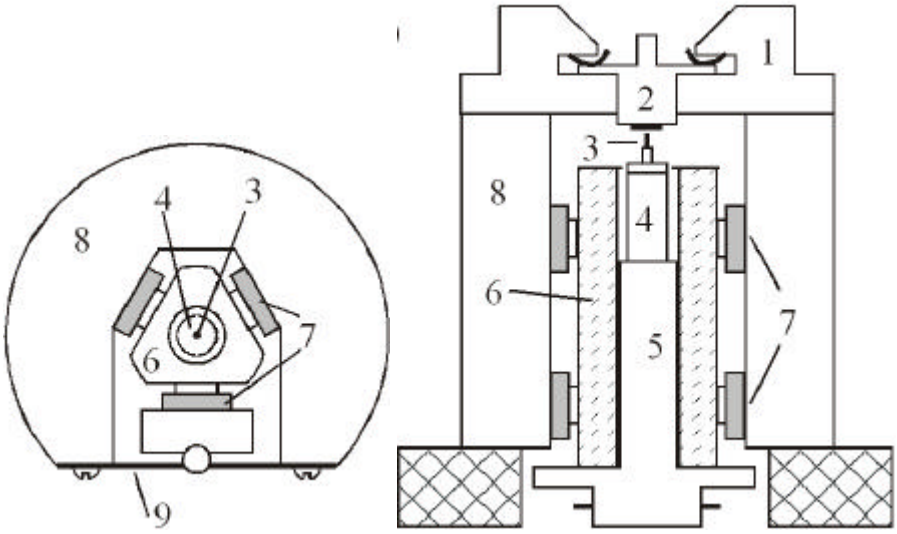
1.2.2 Piezoelectric motors

Piezoelectric motors (piezos) are used to manipulate the position of the STM tip with a high degree of accuracy without significant vibrations or heat. Piezo motors are generally made with ceramic materials which means that they do not have to be magnetically shielded and can be made more compact and efficient than electric motors. Traditional electric motors can not achieve the resolution of fine movements found in a typical piezo motor. The nature of a piezo motor also leaves it free from short circuits, making it more reliable, and able to operate at high energy efficiency, thus reducing any heat load on a cryogenic system.[3]

Piezos use crystal structures that change shape when a voltage is applied. By attaching two sides of the crystals to different objects this can be used for generating a small translation or rotation. Piezo motors can then be stacked to

allow for translations in multiple dimensions. Our piezo motor consists of a single crystal that we can apply voltages in different quadrants to move along any of the three axis. This is the basis of the tip approach and scanning capabilities of the STM.

For larger translation requirements a motor can be built that "walks" the tip closer in a series of small steps. Our setup uses six motors on two levels distributed at equal distances and angles around the tip. The crystals are only attached to the outside of the motor but are able to hold the tip mount through the use of friction. By applying a large voltage to one crystal very quickly the crystal will slip to a extended or sheared position. This is done, in turn, to each piezo until they are all in the new position. If the voltage is then slowly lowered back to zero the piezos will return to the normal state causing the tip mount and tip to move.[4] This setup relies on the difference between sliding and kinetic friction. The tip walk is considered a course movement since it is not as accurate as the normal piezo movement and causes more vibrations but is able to go longer distances. The distance for the walk is only restricted by the mount length that can be used instead of just the normal extension distance. Figure 1.3 shows a diagram of the piezoelectric motor used in the STM and the placement of the six stacks. The course motor is used to approach the surface within a few hundred nanometers before the final approach using just the normal piezo movement.



(a) Top view (b) Side view

Figure 1.3: Piezoelectric motor
The STM head with piezo motor, (a) top view and (b) side view. (1) Sample Receptacle, (2) Sample Holder, (3) Tip, (4) Tube Scanner, (5) Scanner Holder, (6) Sapphire Prism, (7) Shear Piezo Stacks, (8) Macor Body, (9) Spring Plate (not to scale).[4]

1.2.3 Lock-in amplifiers

Lock-in amplifiers are a common tool for improving signal quality by reducing noise. The method used to remove noise from a signal also lets us see the derivatives of a signal. We used several Stanford Research lock-in amplifiers to find the dI/dV curve. Lock-in amplifiers are a form of phase sensitive detectors. They introduce an oscillation to the input signal that is sent to the measuring instrument and then the lock-in amplifier removes any resulting signal that is unaffected by the oscillation.[5] They use either an internal or external tool to modulate the signal sent to the instrument. In the case of a STM this signal is the bias voltage.[4] Using the fact that any two sine waves of different frequencies are orthogonal you can multiply the reference and measurement signals together, then integrate over time to remove most of the noise.[5] To obtain the

first derivative the lock-in finds the component of the detected signal with the modulation frequency which is proportional to the first derivative.[7] A signal oscillating at the second harmonic will be proportional to the second order derivative, and so forth.

CHAPTER 2

NANONIS AND LABVIEW

The Nanonis Software interface was written in LabVIEW and provides an abstract programming interface (API) for custom programs. The API is accessible through LabVIEW, some .NET, and other programming languages. We chose to use LabVIEW directly because it would have a higher performance than interfacing another language through LabVIEW and because several members of our group were already familiar with it. LabVIEW also has the advantage that many other systems, such as our lock-in amplifiers, provide a similar API to interface with. This will let us extend our software further in the future and provide increased flexibility in which tools we can use.

LabVIEW is produced by National Instruments as a companion programming language and development environment for their hardware. It differs from traditional programming languages by focusing on a visual representation of code stored in files called VI. VI serves a dual role in this model, both as a user interface, and as a replacement for traditional functions. A VI is split into two portions: the front panel and the block diagram. The front panel serves as both a user interface and the place where input variables are passed. The block diagram contains all of the internal logic to the VI. "Wires" run between each VI in the block diagram to simulate passing variables and provide a flow to the program. VI can be executed in any order, as long as all the incoming wires have the output from their respective VI outputs, allowing LabVIEW the ability to parallelize some code. The visual code closely mirrors C, which is the language LabVIEW was originally written in, but has extensions for things like objects and dynamic types.

The dual role of VI have some disadvantages that we had to work around. A common software pattern, or best practice is to provide a "thick" data layer to do all the processing and a "thin" user interface that can be easily changed or replaced. By merging both of these, changing the user interface is much more difficult, and in some cases impossible. A VI can not change what items are on the front panel so everything must be populated at compile time and are largely static. Most LabVIEW programs are simpler and do not need dynamic interfaces. The more complicated programs often create a new window for each new interface, but this method becomes confusing and has other issues. To work around this we often used sub-panels and inserted different VI running simultaneously in those sub-panels depending on user input.

Since VI needed to be running to be displayed in a sub-panel, we created an asynchronous process for each display that ran until the program completion. We also used asynchronous processes to take data, and do other actions like tip movement, so they could be done independently of the user interface. This separated the critical processes, like withdrawing the tip and taking data, from interfering with user interface trying to update a graph or process a button press. We also could give higher priority to a particular VI that we deemed important, and slow graphs updates down to update only as fast as the screen could display them. LabVIEW uses a producer/consumer software pattern for most inter-VI communication. One or many VI can add data to a queue that could be retrieved by each VI that had access to that queue. LabVIEW uses this to pass notifications and data between processes. LabVIEW lacks any way of telling the current status of a running process so we created a dedicated process that would always run in the background and use our notification system to tell all of the other VI to terminate when the main program would exit. Our main

VI would send a "keepalive" signal to that process which polls the queue at regular intervals, and sends a signal to exit if the keepalive time-limit has been exceeded.

Because a VI has the only reference to the controls on the front panel it is nearly impossible to reuse any user interface code by abstracting it to a sub-VI as any reference to that control must be passed along as well. This leads to very complicated code which becomes hard to understand and error prone with all the wires that accompany this method. Instead, in each user interface VI we created a while loop with an event poll and created specific events that could be reused for many different purposes. Since LabVIEW lacks callbacks, event polls are needed for most user interface actions anyway and allowed us to produce cleaner code.

LabVIEW added classes long after its initial development so it differs from other object oriented programming languages. The majority of object oriented programming languages create objects in a separate space called "the heap" and pass around pointers to the object in order to reduce memory. This method does not guarantee integrity of class between multiple processes so LabVIEW creates separate copies of the class for each process. Unfortunately, this greatly increases the memory used if the class has a large amount of data, and makes it very hard to communicate between processes while structuring a large program in any meaningful way. LabVIEW does offer a DataValueReference which acts like a pointer, but guarantees atomic access to the class to preserve integrity. However, class inheritance when used with DataValueReference has yet to be implemented, and would cause us to lose one of the major advantages of using a class-based programming structure. National Instruments, instead, suggests

Algorithm 2.1 Graph Resize

Algorithm to resize graphs and charts in LabVIEW in a deterministic manner

1. Turn off panel updates for speed.
2. Find coordinates of plot area and move legend to those coordinates. This assures overall bounds are not affected by the legend position.
3. Find overall bounds and adjust x-axis, y-axis, and plot area bounds to fit inside the overall bounds with correct margins.
4. Adjust plot width and height to appropriate size. This changes the overall bounds, and scales the axis appropriately.
5. Adjust overall position of graph or chart.
6. Move legend back to desired position.
7. Turn panel updates back on.

creating a wrapper class and VIs that checks the object out of a queue, calls the real VI, and then checks the object back into the queue.[1] This preserves data integrity while emulating the functionality of other languages. This significantly increased the number of VIs in the project but made the code much easier to develop and maintain.

Graphs and plots in LabVIEW can be resized and moved programmably, but the behavior is often non-deterministic, resulting in graphs that are cut off or misaligned. We established a specific sequence that allowed us to work around this issue, which we describe in algorithm 2.1.

CHAPTER 3

CODE STRUCTURE

Our custom controller code is divided into one main Controller class and several different utility classes that all support different functionality. The main file used to launch the program is STM.vi. Figure 3.1 shows the basic layout of the STM controller program.

3.1 Controller Class

Unlike every other class, Controller uses a DataValueReference wrapper to synchronize data between processes. Controller requires a configuration file to read in the default settings for any given machine. That file, along with every other configuration file, is accessed using LabVIEW's configuration file API. A configuration file contains multiple sections delimited by a header enclosed in square brackets. Each section has a list of keys on a separate lines, followed by a equal symbol, and then a value. Controller uses those parameters for any settings that are not changed frequently, such as the walker motor pulse settings or the Nanonis install location, and are machine specific. The configuration file also houses a list of channel names and parameters for that machine so they can be customized.

Controller is also used to maintain the state of the system. It maintains a list of the lock-in amplifier configurations, keep-alive status, points to scan; motor, channel, and channel configuration, daily counter, program signal system, and tip status. The daily counter is stored in a configuration file "counter.ini" and will increment each time Get Counter is called, and writes that value back to

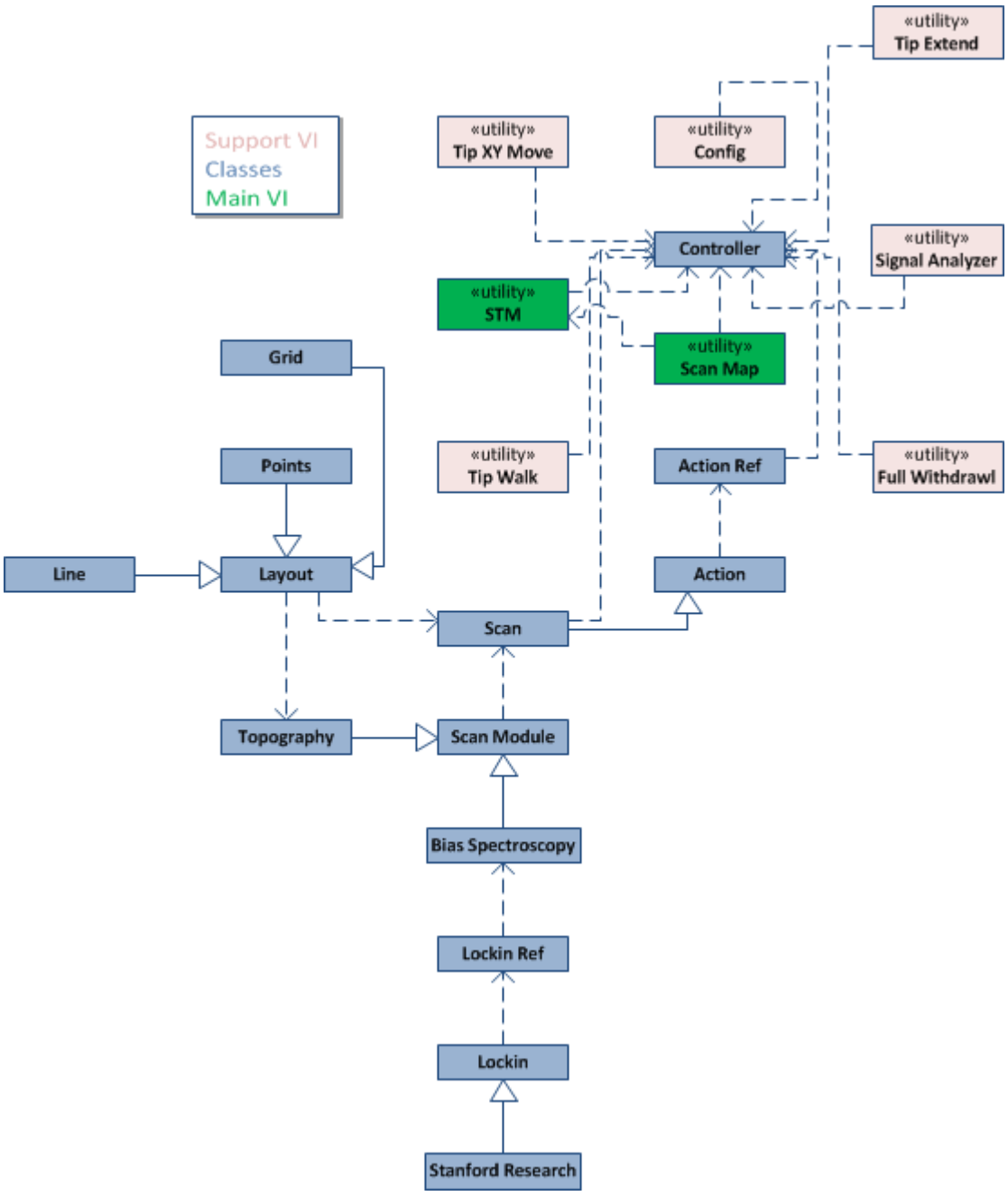


Figure 3.1: STM code model
The layout of the code used in the STM controller program. The solid lines show inheritance between classes and the dashed lines functional dependencies or function calls.

the file. This is used to keep track of any experiments that are run on a given day, and will automatically reset after midnight, local time. The program uses one dedicated LabVIEW notifier to send messages between almost all running actions to communicate events such as the start or finish of a scan. Tip status is Boolean, and can be either extended or not, and is used as a safety check before a map scan can be run. The class also maintains a buffer of averaged channel data that can be used to monitor various tasks. On initialization, Controller will check to see the state of the Nanonis software, launch it if necessary, set any hard coded parameters, and load the remaining default parameters from the configuration file.

We currently hard code the conversion from nanometers to volts for the z-piezo in the Nanonis software so we can view z position as a function of voltage. Our group prefers to view the z distance directly in voltage since it does not require any calibration based on temperature. We also set the oversampling to 1 to give us as close to the unaveraged data as possible. This allows us to use the data from the Nanonis oscilloscope for our signal analyzer (section 3.7). We set the Nanonis z-controller to measure in logarithmic current because it produced a consistent behavior in our testing and is the most common used for STM setups.[10]

The Controller class also contains some utility functions for features not yet supported by Nanonis, but we hope to see in the future. Nanonis currently lacks the ability to directly ramp the bias at a given rate so we wrote a voltage set VI that instead uses the Nanonis Bias Scan tool to ramp the bias without taking any data. We wrote a similar tool to move the z-piezo at a given rate, but that requires version 4.5 of the Nanonis Software which is too advanced for our

hardware. It also contains a tip approach method that limits the speed at which the Nanonis hardware controller will move the tip in the feedback loop (section 3.6).

3.2 Actions

The Action class is used to describe the status of a given process. This allows an event in the user interface to tell another process to exit gracefully instead of terminating in some unknown state. Action is used by directly by VI like Tip Extend and Tip Withdraw that run without much input or output. The Action class is extended by Scan. The Scan class keeps track of the points to scan using the Layout class, what the scan does at each point using the scan module classes, and where any files should be saved.

3.3 Layouts

Layouts are classes used to describe where a scan or other event should take place. They are very simple and not synced between processes so Layouts are one of the only classes that are not mirrored by a reference wrapper. The Layouts class allows new patterns to be added later without adding any code to the scan map except 3D displays. A Layout is responsible to specifying the next *xy* local point to scan, what that point is in global coordinates, and to read/write its configuration from/to a file. Currently there are only three types of Layouts: points, line, and grid. Points refer to a scattered set of points throughout the map. Line is specified by a start and end point and the number of places to scan

along the line in equal units. Grid is specified by a center xy, width and height, number of divisions or pixels along each axis. Grid is also used to specify the display coordinates between 3D maps so it has two functions to translate between local and global coordinates. The transformations are given by equation 3.1.

$$\begin{bmatrix} x_g \\ y_g \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\frac{w}{2} \\ 0 & 1 & -\frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{w}{P_x} & 0 & 0 \\ 0 & \frac{h}{P_y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{P_x}{w} & 0 & 0 \\ 0 & \frac{P_y}{h} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{w}{2} \\ 0 & 1 & \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_g \\ y_g \\ 1 \end{bmatrix} \tag{3.1}$$

3.4 Lock-ins

The Lock-in classes are used to manage all the settings of external and internal lock-in amplifiers used with the controller. Currently only the Stanford Research Lock-in is implemented. The parameters must be edited by hand but we hope to make any configuration automatic through the GPIB connection. Lock-in has the ability to turn a given ripple on and off by adding an external or external signal to the bias channel and outputting it on a separate channel called the bias-calculated channel. The addition of the two signals is done on the Nanonis hardware so the signal is only constrained by the sample rate of the hardware and not the communication speed to the computer. Like Layouts, Lock-ins are responsible for reading and writing to configuration files.

3.5 Scan Modules

Scan Modules are actions that can take place at each point along a map scan. They must create their own configuration and data display user interfaces to display inside the map window. Scan modules are also responsible for read-/writing their configuration and data to the appropriate files in a given directory, as well as providing a run method to be called at each point.

3.5.1 Bias Spectroscopy

The bias spectroscopy is used to wrap the functionality built into the Nanonis controller, as well as provide additional functionality not in the original software. For a single scan we record the topography, move the tip to a new point at $V_{spectroscopy}$ and $I_{spectroscopy}$, slowly ramp the bias while recording the channels, and return to the original state by reversing the same actions. We can also ramp the bias in multiple cycles, both forward and backward, to account for any hysteresis, or track changes over time. Before a scan may be initiated the tip must be on the surface with the Tip Extend method (section 3.6.1), have at least one lock-in amplifier selected (section 3.4), and all the parameters set within their given limits. The procedure for a scan point is given in algorithm 3.1, and figure 3.2 shows a simplified model of the process. The figure was generated from a test script with test parameters chosen to exaggerate some steps so their features were more visible. The data is saved in a custom configuration with the parameters for the scan as ASCII, followed by a null byte, and then the data in a binary format. An example Matlab script for reading the data is given in appendix A.

The bias spectroscopy configuration setup is designed to be as flexible as possible, with some measures to prevent the tip from crashing. A scan may be given a description and loaded from a configuration file. The first thing a user will be asked to determine is the spectroscopy set point and voltages with the same topography settings used in the main window. The user may then select channels to record, however the lock-in amplifier and current channels will always be recorded no matter what is selected. The user may then specify the "*Start delay*" and time to "*Establish feedback*" used in algorithm 3.1. The proportional and integral (PID) settings by default mirror the ones used for topography scans, but may be altered. The three speeds used in the point scan may then be altered but must be positive, finite values. A scan may be run for any number of cycles, and may also scan when returning from the final voltage to the initial voltage with a delay in between each cycle. The scan cycle is specified from segments of starting voltages to ending voltages with a given number of points in a given segment. This is one of the significant advantages over the former system because Nanonis allows ranges of energies to be studied in great detail without having to scan points in less interesting regions. The final parameter the user must select is the locking used in the scan to provide the ripple.

Algorithm 3.1 Bias spectroscopy

(after arriving at the new point with feedback on for topography parameters)

1. Record the z and current values at the topography voltage and set point
2. Ramp the tip (with speed "*Feedback*" V/s) with the feedback on to the voltage $V_{spectrography}$ and set point $I_{spectrography}$ for the spectroscopy scan
3. Delay *Establish Feedback* ms to establish a stable tip with the feedback cycle
4. Set the appropriate proportional and integral values
5. Record the z and current values at the spectroscopy voltage and set point
6. Turn the feedback off
7. Ramp to the starting voltage (with speed "*Start Ramp*" V/s)
8. Wait for "*Start Delay*" ms
9. Turn on the Lock-in ripple
10. Call Nanonis Bias Spectroscopy Scan
    (a) Wait "*Before Sample*" ms
    (b) Record channel data for "*Average time per point*" ms and average values
    (c) Ramp bias to next value (with speed "*Sample point*" V/s)
    (d) Repeat from 10 until the last point in the cycle is reached
11. Repeat 10 for backwards and repeated cycles
12. Turn off the lock-in amplifier ripple
13. Ramp back to starting voltage (with speed "*Start Ramp*" V/s)
14. Ramp back to spectroscopy point (with speed "*Feedback*" V/s)
15. Turn on feedback with settings for topography
16. Wait for feedback to be established

3.5.2 Topography

While Topography is implemented as a Scan Module, it is currently not intended to be run at a single point. Nanonis has a dedicated topography scan for a grid that is much quicker than doing a full map scan. Topography pro-

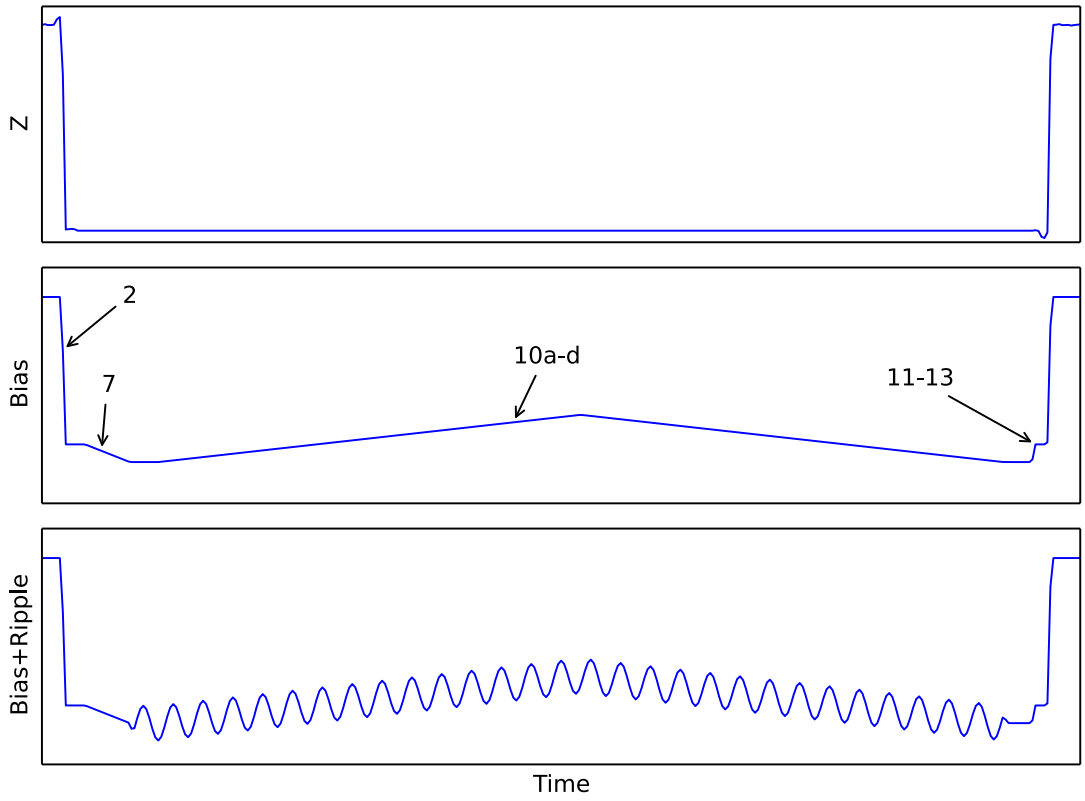


Figure 3.2: Bias Spectroscopy
A cartoon of the Bias and Z position of the tip for a bias spectroscopy scan at a single point. The numbers correspond to the algorithm listed in 3.5.1. The plot was generated using the bias spectroscopy scan module with test parameters chosen to exaggerate some steps so their features were more visible.

vides a "Run Nanonis Topography.vi" method for calling that function outside of a map scan. Topography stores both the current and z data for the Nanonis scan, as well as calculates the minimum, maximum, mean, standard deviation for each to let the user adjust the display accordingly.

3.5.2.1 Topography Display

The topography user interface shows two graphs that display the z and current of a scan. The scan and display processes communicate using a separate

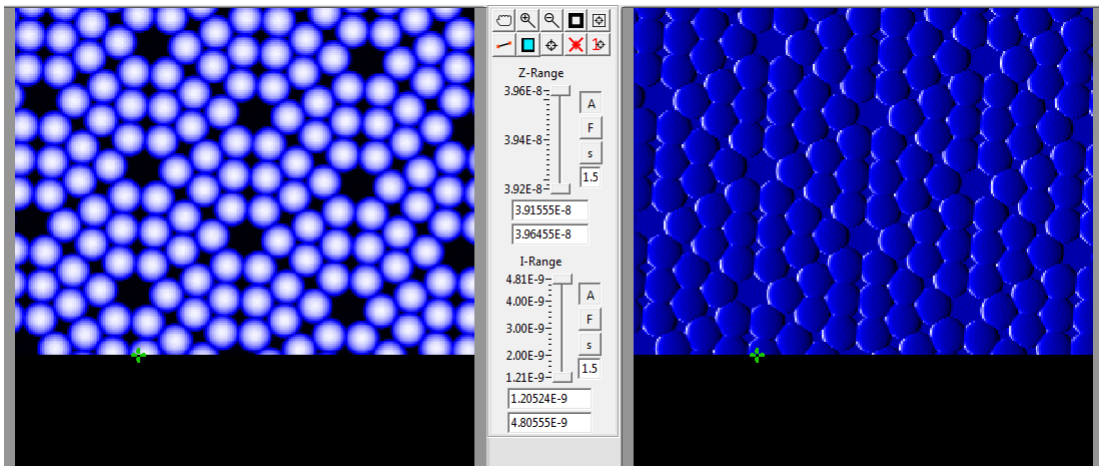


Figure 3.3: Topography Display
The topography display mid-scan for simulation data.

LabVIEW notification system to avoid all of the other processes being forced to utilize extra processing cycles. When the Scan Module Run VI receives any data it sends a notification to the display which updates the graphs. The graphs can be configured to display any z-scale within the range of data, scaled to a range based on the standard deviation, or auto-scaled for optimal contrast by LabVIEW. The graphs use a separate local coordinate system based on the grid layout so a scan will appear unrotated and scaled to fit. The Topography display can also generate a new point of view, line scan, or set of points to scan based on the relative coordinates. It can also update the xy move feature so the tip may be manipulated in the local coordinates. The first cursor (green) is reserved for the real-time tip location, the next two cursors (yellow) for line and grid selection, while the rest of the cursors (red) are used to display the map scan points. Figure 3.3 shows the topography display for a typical scan using simulation data.

3.6 Tip Approach

There are two methods for approaching the surface of a sample with the piezoelectric motors: Tip Walk and Tip Extend. Tip Walk is a course search for the surface used for traveling larger distances. Tip Extend is used for the a slower approach to the surface and is used by the Tip Walk for searching within the course intervals, as well as approaching after the tip is close to the surface.

3.6.1 Tip Extend

Locating the surface of a sample is one of the most basic functions for a STM, but requires careful execution to avoid crashing the tip into the surface and destroying it. To locate the surface from an unknown distance we slowly extend the tip towards the surface until we detect an increase in the current from electrons beginning to tunnel between the tip and the surface. Because communication with the Nanonis controller is limited to a few hundred Hz, the normal method of extending the tip at a constant rate and constantly checking the current would not be fast enough to assure a safe approach. Our tip approach method uses the feedback loop of the Nanonis controller, as this will prevent surface contact and measurements are preformed on the real-time operating system. To limit the speed of the approach we modify the PID settings for the approach then reset the values after the surface is reached. Nanonis PID excludes the differential component, and uses a time constant component (Tc) that is related to the integral (I) by $I = \frac{1}{Tc}$. This means the piezo will move at a given speed $v = P(1 + I) = P\left(1 + \frac{1}{Tc}\right)$.[10]

We ran a mapping program over various P and Tc to see if we could find an appropriate function to calculate the velocity for a given I and Tc. We found that the velocity for a given PID would change greatly when we altered the calibration (A/V) of the current channel and varied more slowly if we changed the bias or the set point current. We observed for low values of P the tip would begin to oscillate while at very high values the tip would just hover at the withdraw point. For a current channel calibration of $1nA/V$ we chose $P = 8.33 \times 10^{-13}m$. We then ran our mapping program for Tc between $10^5 s$ and $1s$. To account for differing orders of magnitude and areas of interest we ran over three different arrangements of points: logarithmic, linear, and exponential. We then fit $v(P, Tc) = aP\left(1 + \frac{1}{bTc}\right)$ for a and b to the map. Figure 3.4 shows our fit function and mapped data. We observed oscillations in velocity for $Tc < 2 \times 10^{-4} s$ so we only fit data above this threshold. We also observed an increase in the error ratio above 4×10^{-2}, but we believe that this was an artifact of the mapping program and can be ignored. The data above 4×10^{-2} was included in our fit.

We use the fitted data to calculate an initial velocity for our approach. In order to account for the small variations in velocity when we change the bias or set point, we monitor the approach and adjust the Tc based on the observed speed. The velocity is increased by recalculating Tc for the expected velocity by multiplying it by factor of 1.01 for any observed velocity less than 85% of the desired velocity. Similarly, we decrease the expected velocity calculation by a factor of .9 for any velocity greater than the desired velocity. This assures us that we approach between .85 to 1 times the desired velocity while erring on the side of a slower approach. We also weight the observed velocity by averaging the last 10 observations to prevent any sudden spikes or oscillations. As the tip approaches the surface the feedback loop will automatically decrease the

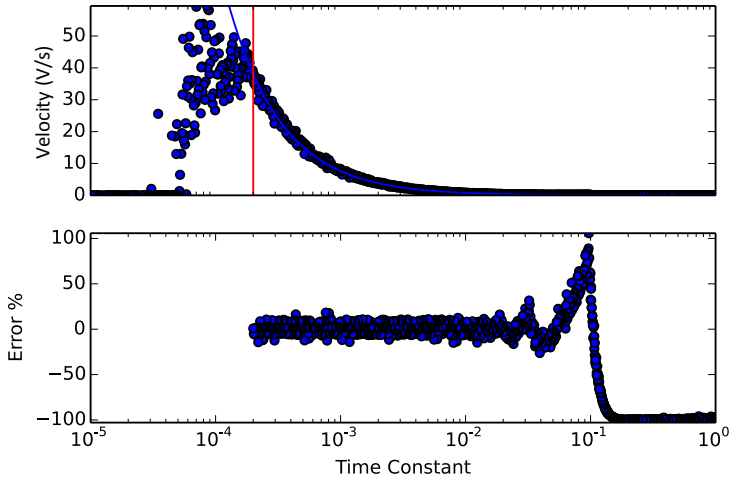


Figure 3.4: Mapping of time constant and integral to velocity
The observed velocity for $P = 8.33 \times 10^{-13} m$ and varying Tc along with fit function (Top). Data below the cutoff (vertical red line) was not included since the behavior of the tip became unstable and began to oscillate. (Bottom) Error of fit function compared to data as a percentage. The large increase in error towards higher Tc is likely an artifact of our mapping program rather than a sudden change in velocity observed.

speed. To avoid altering this effect we monitor the current and do not make any changes to the PID if the current increases or decreases by more than 10%. Figure 3.5 shows a tip approaching the surface and the speed adjusting, first to the expected range, then as the tip nears the surface and the current increases.

3.6.2 Tip Walk

The tip walk method is used to find the surface of a sample from a much greater distance the first time after sample is added. Our method begins with a fully withdrawn tip that we extend at a pre-configured rate using the normal tip extend. If the surface is found we withdraw the tip and repeat the measurement, typically only a second time, to avoid any false positives. The number of cycles

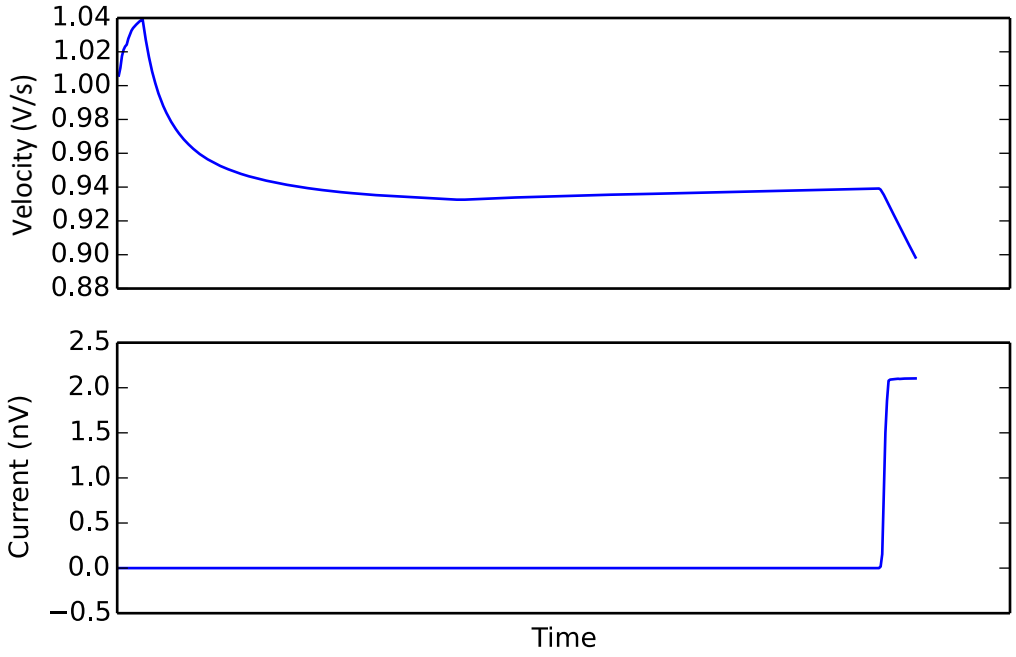


Figure 3.5: Velocity and current observed for feedback approach method
We measured the current and velocity for a tip approach using the feedback approach method. The desired velocity was $1V/s$ and was done using the Nanonis simulation software, due to the STM not being connected, but with the PID fit observed from our controller. The sharp drop in velocity at the end corresponds with the tip approaching the surface and the resulting increase in observed current.

can be configured as well. If the surface is not found we fully withdraw the tip and send a per-configured pulse to the piezo motor walker. Figure 3.6 shows a typical cycle for the z position and motor driver channels where no surface was found.

3.7 Signal Analyzer

The signal analyzer is used to display an input, output or internal channel in the time or frequency domain. This can be used to check for signals in the noise

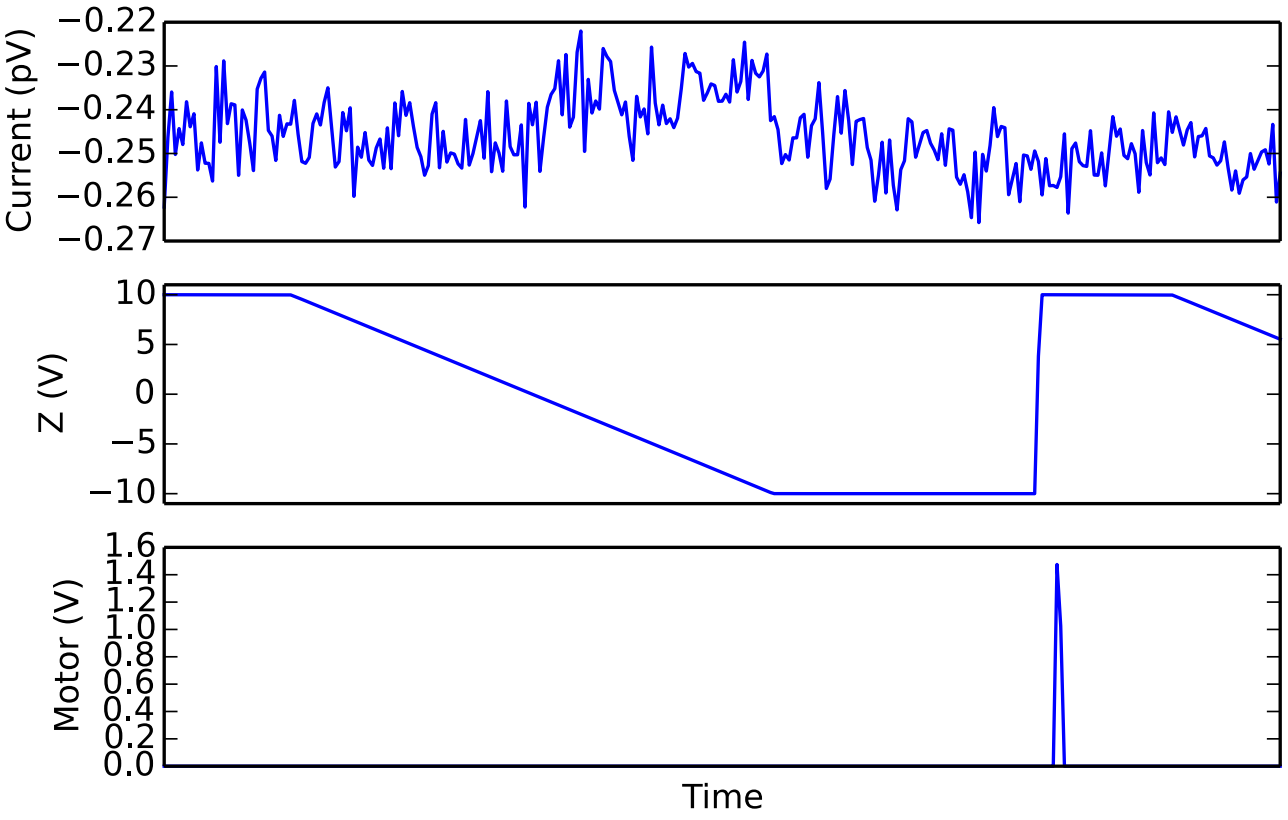


Figure 3.6: Tip Walk
We measured the current, z position, and motor channel for a single cycle of the tip walk approach. The current channel only detected noise so the motor was pulsed and the tip extend cycle starts again.

or diagnose the STM in general. By design, Nanonis does not store unaveraged data and the API only has access to averaged data, therefore we pull the data from the graph in the Oscilloscope 2T module. We can not access more data than what is on the graph and there is no guarantee of the sample rate so we can not concatenate multiple samples without possibly overlapping data. The Oscilloscope 2T data is displayed in the time domain mode of the signal analyzer display. At periodic intervals the data is queried, new mean and standard deviations are calculated, and if the frequency mode is active the Fourier trans-

form is calculated and displayed. The Fourier transform uses LabVIEW's FFT Power Spectrum and PSD VI, and has the weighting and windowing options as controls for easy access. Figure 3.7 shows the two modes of the signal analyzer. The graphs use algorithm 2.1 to switch between modes.

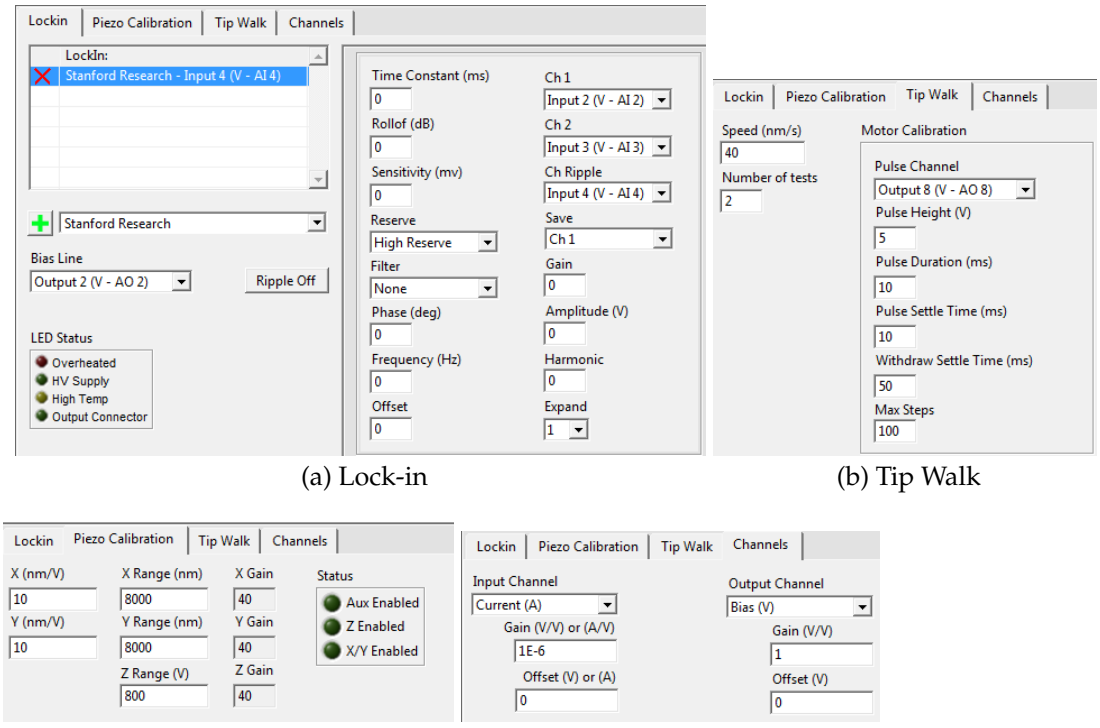
3.8 Configuration

For parameters that do not need to be changed as often we provided a separate VI to expose them to a user. Figure 3.8 shows the configuration display. The Lock-in tab allows multiple lock-in amplifiers to be added, as well as temporally turning the ripple on and off for testing (section 3.4). Piezo calibration is used to set the range and conversion between nanometers and volts. Tip Walk displays the parameters specific to the tip walk approach including the pulse parameters needed to activate the motor (section 3.6.2). Channels allow the configuration of input and output channel offset and calibrations.

3.9 Scan Map

Scan map is used to iterate through various points in space, and execute a scan module action at each point. The scan map uses the various Layouts (section 3.3) points, line, and grid to structure the scan. The scan modules can be stacked so that several are run in succession at each point. However, only the Bias Spectroscopy module is currently implemented. As a safety precaution the scan map VI verifies that each module added is correctly configured before a run is allowed. The run scan process is then called in a separate thread and interacts

Figure 3.7: Signal Analyzer Display
Signal analyzer display. (Top) time domain (bottom) frequency domain.



(a) Lock-in (b) Tip Walk

(c) Piezo Calibration (d) Channels

Figure 3.8: Configuration Display

Configuration display. (3.8a) is used to add and configure the lock-in amplifiers, (3.8b) is used to configure the settings used in the tip walk (section 3.6.2), (3.8c) is used to change the piezo *xy* range and distance traveled per volt, (3.8d) is used to change the gain and offset for the input and output channels. As (3.8a) is the first tab seen when the configure VI is shown, it also displays the system status.

through the normal messaging callbacks. Any module shown should have configuration and display VIs that are automatically added to the Scan Map VI when they are added to the scan. A run is executed in three stages: the initialize stage where the tip is not moved but classes may be initiated and files created, the run stage where the tip is moved, and the end stage that allows each module to take care of any clean up required.

CHAPTER 4

PROCEDURES AND RESULTS

4.1 Workflow and tip safety

When the scanning tunneling microscope controller program is started it will check the status of the Nanonis software. If it is not running it will launch the program and continue to wait until the program has loaded before allowing any user interaction. If a user wants to change the gain of the system it should be done in the beginning since there is no way to tell if a button on the system is changed without constantly polling. The system loads all of the settings from several default file configurations that can be found within the Defaults directory. The files used by the software have the ".ini" file extension and have key value pairs separated by an equal sign. Figure 4.1 shows the main window. Configurations can also be saved and loaded for the topography and bias spectroscopy settings. To access the more commonly used configurations the user can click the "Config" button which will pull up a menu of the lock-in amplifier, piezo, tip walk, and channel settings (section 3.8).

For a new tip or sample, the tip must first be walked to the surface using the Tip Walk function. At the end of the tip walk the tip is automatically withdrawn. (section 3.6.2). To reach the surface again the user can use the tip extend method (section 3.6.1). A topography scan can be done while at the surface in feedback mode or at any fixed height (section 3.5.2). The user can use the grid settings to move the scanning grid to a new location, either by pixel or by distance. The translation is measured in the local coordinates of the last grid scan. However, neither the tip or local coordinate system will change until a scan is

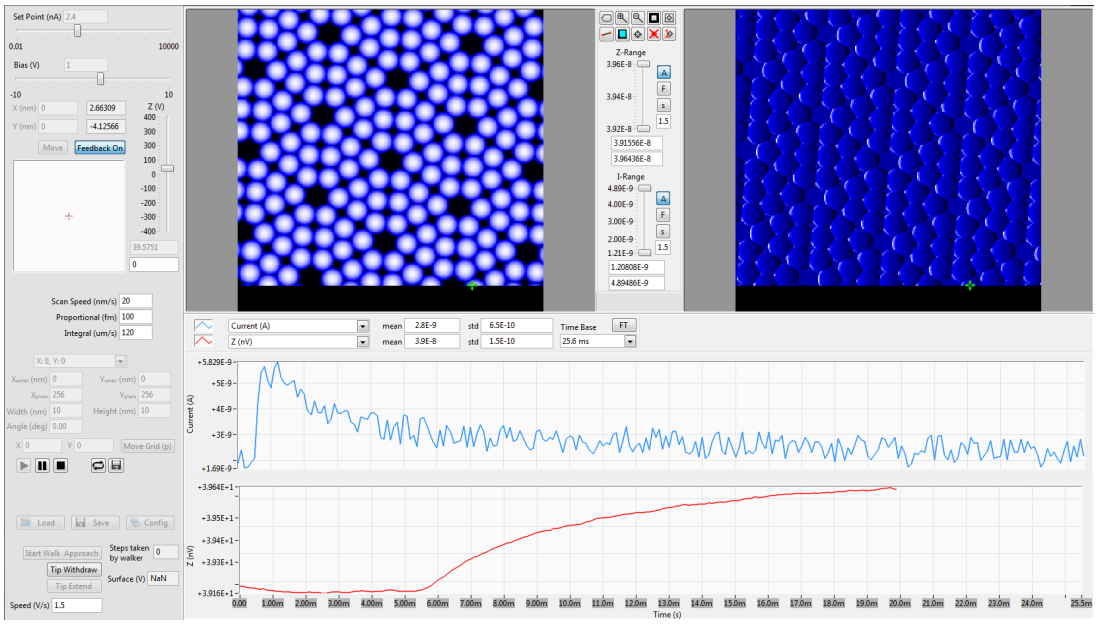


Figure 4.1: Davis STM software main window
The main window of the STM program. On the left side are the controls for bias voltage, set point, tip movement, a graph showing the tip position, PID settings, scan speed, local grid, topography scan actions, and tip approach and withdraw functions. The top right is the topography display and the bottom right is the signal analyzer display.

topography or map scan is initiated. The user can also move the tip in the x, y, and z directions. The x and y directions are again measured in local coordinates, the movement is done at scan speed, and only initiated once the user selects the move button. The global grid, located below the xy coordinates can be selected to locate a region in the global coordinates and translate them to the local coordinates for a movement. The global grid also shows the current grid scan location within the global coordinates, though if the global area is too large this may not be distinct. If the user changes the z coordinate the feedback is automatically turned off to account for the desired change in position. The change in z position is nearly instantaneous.

After an appropriate area has been located, the user can execute a map scan (section 3.9). Figure 4.2 shows the scan map window. One or more modules can

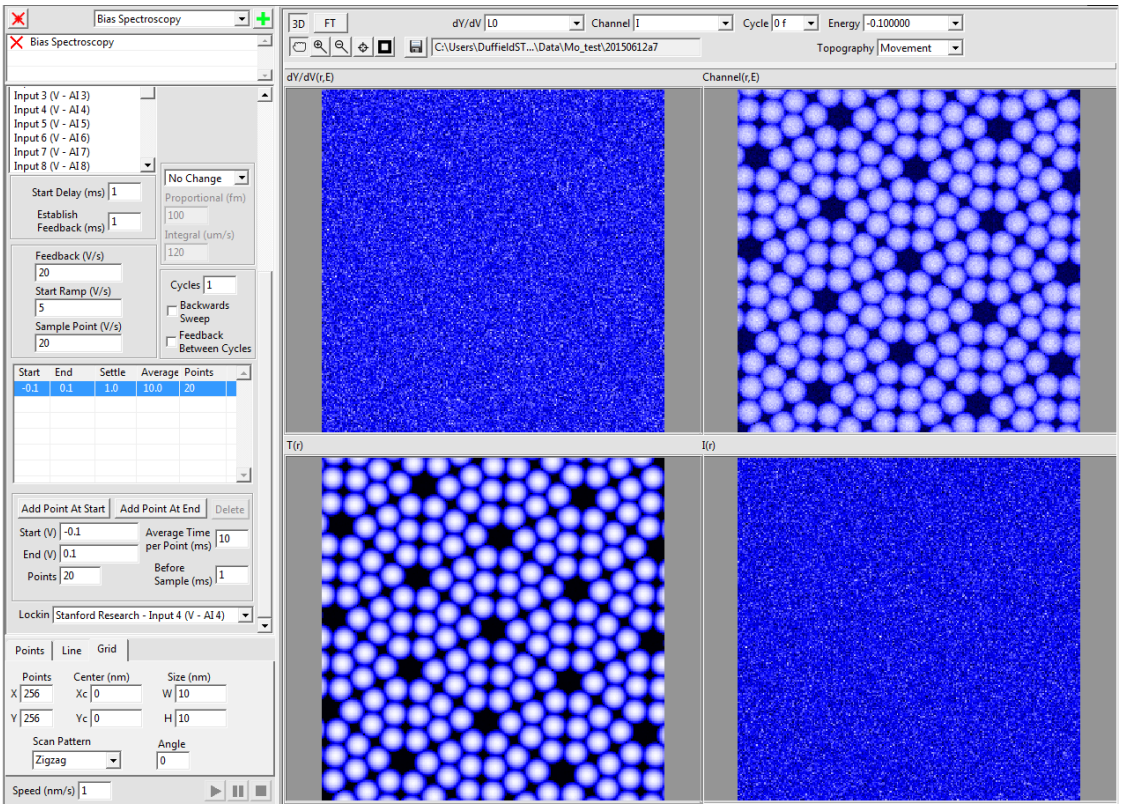


Figure 4.2: Davis STM software scan map window
The scan map window. The top left side provides the configuration for the current scan modules (bias spectroscopy shown). The bottom left is used to select which layout should be scanned and the scan actions. The right side shows the current scan module display.

be added to the scan and each will be executed at each point in the scan. To decrease the chances of the tip crashing into the surface or the user disrupting the current scan, several safety measures were implemented that prevent changes to a setting that would impact the current function. A list of all the fields that are locked and enabled for each state of the program is given in table 4.1. There is no programming state that may block the tip withdraw since it should be accessible for any problems that happen during the program run. Tip withdraw will also stop any ongoing scans in progress.

	Topography Scan	Scan map	Configuration mode	Tip Extend/Walk	Tip Withdraw
editable items	proportional/integral scan speed		feedback move x, y, and z current grid scan speed proportional/integral	scan speed	nothing
locked items	feedback move x, y, and z set point and bias configuration mode current grid scan map play	feedback move x, y, and z set point and bias configuration mode current grid proportional/integral topography play scan speed	topography play scan map play	feedback move x, y, and z set point and bias configuration mode current grid proportional/integral topography play scan map play	everything

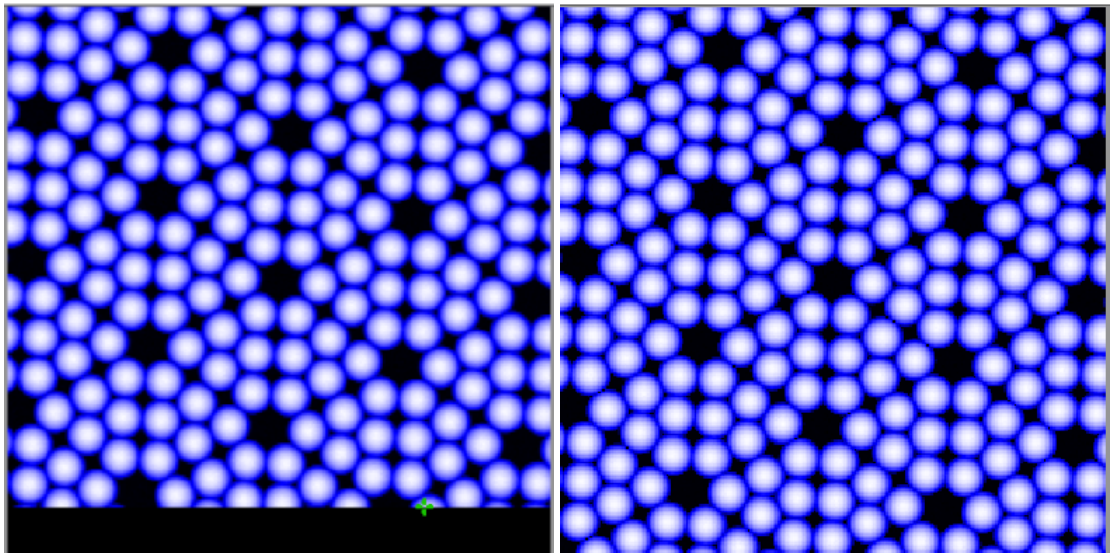
Table 4.1: Fields that are locked and editable for each state of the running program.

4.2 Results

Due to the current hardware construction and ongoing projects of the current STMs most of our results were obtained through simulated data or with the isolated electronic systems. We were able to benchmark our system output and input against an external oscilloscope and our Stanford Research lock-in amplifier. The input/output voltages and currents agreed to within the precision of the testing parameters. We were then able to use the Nanonis system as a tool to measure its own input and output though connections between various ports. There are three areas we focused our diagnostics on, the tip movement, topography, and bias spectrography.

The tip movement is controlled through four output ports, two for in-plane movements, one for slow z-moment with the piezo, and the final output that can send a pulse to the piezo course motor. Each was benchmarked against the oscilloscope to assure the correct voltage. Figures 3.6 and 3.6.1 show the walk and tip approach data respectively. The tip walk shows the trajectory of the tip as it is extended and then the pre-configured motor pulse when the tip has been fully extended and retracted. The tip approach demonstrates a gradual approach to the surface without any great increase or decease in speed until it nears the surface. Both these behaviors were used to confirm the algorithms used in each process were working as intended.

The topography was checked using the simulated data from the Nanonis program. Figure 4.3 shows two different topography algorithms with nearly identical results. The first method was obtained using the Nanonis built in topography scan functionality. The second method was obtained from the map



(a) Nanonis algorithm (b) Map scan algorithm

Figure 4.3: Topography benchmark
Topography of simulated data using Nanonis built in algorithm and map scan algorithm.

scan using the same scanning parameters as the first. Both sets of data are processed through LabView and shown in the program during the respective scan.

The bias spectrography was harder to test due to our inability to use the external lock-in with simulated data. We used a custom setup to test the bias spectrography algorithm using several of the unused ports to add additional signals to the system input. Figure 3.2 shows actual data from our system with exaggerated values so it was easy to diagnose. We were unable to benchmark the recorded data against the Nanonis simulated scan without implementing new functionality. The bias spectrography algorithm will be tested against known samples when our system is fully integrated into one of the working STMs.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The scanning tunneling microscope interface represents a significant upgrade to our current systems. The Nanonis commercial hardware was used to replace 23+ year-old hardware and should allow for easier upgrades in the future. The Nanonis software will afford new flexibility with its ability to scan custom energy ranges. Our software interface will further enhance the the Nanonis software by providing a workflow with safety checks for tip and sample preservation, along with further options for custom scans and mapping layouts.

Our software layout is designed to be module, to facilitate easy debugging and upgrades to the system. To work within the limits of LabVIEW we implemented a class system that is wrapped to create references for data sharing, a multi-process architecture, and a communication system to communicate between processes. This allowed us to fit all functionality within two windows for easier navigation than the original Nanonis software which used a new window for each configuration, operation, and display. Our system provides interfaces for configuring the Nanonis hardware, topography and bias spectroscopy scans, mapping scans, signal analysis, and tip control. For the initial implementation we focused on the topography and bias spectroscopy scans, as they were the most crucial to our group. We also focused on the method of tip approach as this is one of the most important parts to tip preservation, and thus scan quality. Our system also provides an architecture that makes it easier to add new classes of experiments which will be essential in the next generation STMs.

As of the time of writing, our system has been mostly tested using the simulation data from the Nanonis simulation program. This has facilitated quicker,

easier programming and testing as it produced a reliable set of data without the worry of damaging an actual tip or cooling our system. The Nanonis hardware and software was further tested using a second STM, as well as input from and output to a Stanford Research lock-in amplifier. We are planning on using the Nanonis system first with a new STM that is currently under construction then moving to upgrade the other STMs. As this system is still being built, the next step for this project is to test with the the actual hardware as it is built.

There are several programming functions that could use further refinement, but were not done due to time constraints. The Stanford Research Module currently relies on user input for computing values. The Stanford Research lock-in amplifier has a GPIB connection so this module should be updated to detect it automatically, add it to the current configuration, update these values from the lock-in automatically. During the course of testing we needed to run an overnight test to record 8 channels of input to diagnose a problem with one of the current STMs. This and other diagnostics should be integrated into the software for ease-of-use and to avoid duplicating work each time we need to run such a test. Finally, while the software can currently run and save data, it can not resume from a computer or LabVIEW crash, or load the existing data when such an event occurs. Much of the functionality for this was implemented but it was not finished and/or tested so we did not make it accessible. It will also be useful to add other tests over time for additional functionality such as z-spectroscopy, and as the system is designed with this in mind this should be relatively straight forward.

BIBLIOGRAPHY

[1] Labview object-oriented programming faq.

[2] G. Binnig, H. Rohrer, Ch. Gerber, and E. Weibel. Surface Studies by Scanning Tunneling Microscopy. *Phys. Rev. Lett.*, 49(1):57–61, July 1982.

[3] Walter. Heywang and Karl.A2 Wersing Lubitz, Wolfram.A3 PY 2008. *Piezoelectricity: Evolution and Future of a Technology*. 2008.

[4] E. Hudson. *Investigating High-TC Superconductivity on the Atomic Scale by Scanning Tunneling Microscopy*. PhD thesis, UNIVERSITY OF CALIFORNIA AT BERKELEY, 1999.

[5] M. L. Meade. *Lock-in Amplifiers: Principles and Applications*. e-edition edition, 2013.

[6] L. Petersen, Ph. Hofmann, E. W. Plummer, and F. Besenbacher. Fourier transform-stm: determining the surface fermi contour. *Journal of Electron Spectroscopy and Related Phenomena*, 109(1–2):97–115, August 2000.

[7] Daryl W. Preston and Eric R. Dietz. *The Art of Experimental Physics*. Wiley, New York, 1 edition edition, January 1991.

[8] M. G. Ramchandani. Energy band structure of gold. *Journal of Physics C: Solid State Physics*, 3(1S):S1, 1970.

[9] SPECS. *Signal Conversion SC5 User Manual*, r3800 edition, May 2013.

[10] SPECS, R4700. *Nanonis SPM Control Software Documentation*, 06 2014.

[11] Andrei L. Tchougreeff and Roald Hoffmann. Charge and spin density waves in the electronic structure of graphite: application to analysis of STM images. *The Journal of Physical Chemistry*, 96(22):8993–8998, 1992.

[12] D. R. Vij. Handbook of applied solid state spectroscopy. 2006.

APPENDIX A

MATLAB CODE FOR SPECTROSCOPY DATA

```
filename = ''; % Change to the appropriate file

%.MSZ - Move Setpoint Topopography
%.MSI - Move Setpoint Current
%.SSZ - Spectroscopy Setpoint Topography
%.SSI - Spectroscopy Setpoint Current
%.* - <channel> <[f]orward/[b]ackward><cycle #>

% try to open file
fid = fopen(filename, 'r');
if fid == -1
error(['Error Opening ',filename]);
end

% read in header
header = '';
byte = fread(fid, 1);

while byte ~= 0
    header = [header, byte]; %#ok<AGROW>
    byte = fread(fid, 1);
end
header = sprintf(header);

% read data
data = fread(fid, Inf, 'float32');

% close file
fclose(fid);

% calculate how long each row of data is
[pathstr,name,ext] = fileparts(filename);

% topo files
if strcmpi(ext, '.msi') || strcmpi(ext, '.msz')
    datalength = 1;
elseif strcmpi(ext, '.ssi') || strcmpi(ext, '.ssz') || 1
    % For channels?
    %SamplePoint
    Key = 'SamplePoint = ';
    Index = strfind(header, Key);
    datalength = sscanf(header(Index(1) + length(Key):end),
                        '%g', 1);
end
```

```
% format data into [x,y,data]
x = data(1:(datalength+2):end);
y = data(2:datalength+2:end);
data([1:datalength+2:end 2:datalength+2:end]) = [];

energy_index = 1; % Change me for different energy levels

% plot data
s = ones(1,length(x))*100;
z = data(energy_index:datalength:end);
tri = delaunay(x,y);
h = trisurf(tri, x, y, z);
% make it look nice
hold on
lighting phong
shading interp
colorbar EastOutside
set(gcf, 'colormap', [linspace(0,1,64);
    linspace(0,1,64); linspace(1,1,64)]');
hold off
view(0,90)
```

APPENDIX B

FILES

The files used in the STM program. There are approximately 330 VI files.

```
counter.ini
Full withdrawl.vi
Nanonis-Session.ini
Poezo Config.vi
Prompt User for String.vi
Run Scan.vi
Signal Analyzer.vi
STM Map Scan.vi
STM.aliases
STM.lvlps
STM.lvproj
STM.vi
temp.ini
Tip Extend Manual.vi
Tip Extend.vi
Tip Extend2.vi
Tip Walk.vi
Wait For Exit.vi

Action\Action.lvclass
Action\Get Notifier.vi
Action Ref\Action Ref.lvclass
Action Ref\Checkin Action.vi
Action Ref\Checkout Action.vi
Action Ref\Get Notifier.vi
Action Ref\Init.vi
Action Ref\Set refQueue.vi

Bias Spectroscopy Scan\Bias Spectroscopy Config.vi
Bias Spectroscopy Scan\Bias Spectroscopy Display.vi
Bias Spectroscopy Scan\Bias Spectroscopy Scan.lvclass
Bias Spectroscopy Scan\Create Bais Spectrpscopy.vi
Bias Spectroscopy Scan\Create Files.vi
Bias Spectroscopy Scan\Get Channels To Record.vi
Bias Spectroscopy Scan\Get Config Vi.vi
Bias Spectroscopy Scan\Get Display VI.vi
Bias Spectroscopy Scan\Get Energy Values.vi
Bias Spectroscopy Scan\Get Name.vi
Bias Spectroscopy Scan\Get Ready.vi
Bias Spectroscopy Scan\Read Config.vi
Bias Spectroscopy Scan\Run.vi
```

```
Bias Spectroscopy Scan\Save Data.vi
Bias Spectroscopy Scan\Save.vi
Bias Spectroscopy Scan\Set Lockin.vi
Bias Spectroscopy Scan\Update.vi
Bias Spectroscopy Scan\Write Config.vi

Controller\Add Lockin.vi
Controller\Add Scan Module.vi
Controller\Add Scan Point.vi
Controller\Aquire Run Lock If Free.vi
Controller\Aquire Run Lock.vi
Controller\Config.vi
Controller\Controller Closed.vi
Controller\Controller.lvclass
Controller\Create.vi
Controller\Delete All Scan Points.vi
Controller\Delete Lockin.vi
Controller\Delete Scan Module.vi
Controller\Delete Scan Point.vi
Controller\Get Nanonis Controller API.vi
Controller\Get Approach.vi
Controller\Get Average Current.vi
Controller\Get Bias Calculated Channel.vi
Controller\Get Bias Channel.vi
Controller\Get Channel Names.vi
Controller\Get Config Mode.vi
Controller\Get Config Value boolean.vi
Controller\Get Config Value Double.vi
Controller\Get Config Value I32.vi
Controller\Get Config Value Path.vi
Controller\Get Config Value String.vi
Controller\Get Config Value U32.vi
Controller\Get Config Value.vi
Controller\Get Counter.vi
Controller\Get Current Channel.vi
Controller\Get Current Grid.vi
Controller\Get Data Length.vi
Controller\Get Extended.vi
Controller\Get Halt Status.vi
Controller\Get Input Channels.vi
Controller\Get Internal Channels.vi
Controller\Get Last Location.vi
Controller\Get Lockin.vi
Controller\Get Lockins.vi
Controller\Get Motor.vi
Controller\Get Nanonis Controller.vi
```

Controller\Get Number of Lockins.vi
Controller\Get Number of Scan Modules.vi
Controller\Get Number of Scan Points.vi
Controller\Get Output Channels.vi
Controller\Get PID.vi
Controller\Get Piezo Calibration.vi
Controller\Get Piezo Position.vi
Controller\Get Piezo Range.vi
Controller\Get Scan Module.vi
Controller\Get Scan Point.vi
Controller\Get Scan Points.vi
Controller\Get Scan Ref.vi
Controller\Get Signal Data History.vi
Controller\Get Signals Notifier.vi
Controller\Get Topo Settings.vi
Controller\Get Walker Max Position.vi
Controller\Get Walker Position.vi
Controller\Get Walker.vi
Controller\Get XYZ Channels.vi
Controller\IV and IS Spectrocopy.vi
Controller\Monitor Signals.vi
Controller\ObtainDataQueue.vi
Controller\Pulse Piezo.vi
Controller\Pulse Walker.vi
Controller\Read Config.vi
Controller\Read Current Z-Piezo.vi
Controller\Read Max Z-Piezo.vi
Controller\Register Windows.vi
Controller\Release Run Lock.vi
Controller\ReleaseDataQueue.vi
Controller\Run Lock Should Exit.vi
Controller\Send Keepalive.vi
Controller\Set Approach.vi
Controller\Set Bias Calculated Channel.vi
Controller\Set Config Mode.vi
Controller\Set Current Grid.vi
Controller\Set Extended.vi
Controller\Set Halt Status.vi
Controller\Set Last Location.vi
Controller\Set Motor.vi
Controller\Set Nanonis Controller.vi
Controller\Set PID.vi
Controller\Set Piezo Calibration.vi
Controller\Set Piezo Position.vi
Controller\Set Piezo Range.vi
Controller\Set Ripple.vi

Controller\Set Scan Ref.vi
Controller\Set Topo Settings.vi
Controller\Set Walker.vi
Controller\STM.vi
Controller\Tip XY Move.vi
Controller\Voltage Set - manual.vi
Controller\Voltage Set2.vi
Controller\Z-Piezo Full Withdraw.vi
Controller\Z-Pos Set - manual.vi
Controller\Z-Pos Set2.vi
Controller\Scan Module Ref\Run.vi
Controller\Scan Module Ref\Save.vi
Controller\Scan Module Ref\Update.vi

Controls\2D-3D.ctl
Controls\Add.ctl
Controls\Borderless Cluster.ctl
Controls\Crosshairs 1x.ctl
Controls\Crosshairs -box.ctl
Controls\Crosshairs.ctl
Controls\Delete All.ctl
Controls\Delete.ctl
Controls\Intensity Graph.ctl
Controls\Line Select.ctl
Controls\Pan Hand.ctl
Controls\Pause button.ctl
Controls\Play button.ctl
Controls\Repeat button.ctl
Controls\Square Select.ctl
Controls\square.ctl
Controls\Stop button.ctl
Controls\Zoom In.ctl
Controls\Zoom Out.ctl

Defaults\Bias Spectroscopy.ini
Defaults\Machine Config.ini
Defaults\Scan.ini
Defaults\Topography.ini

Grid\Get Grid.vi
Grid\Get Max Number of Points.vi
Grid\Get Max XY.vi
Grid\Get Name.vi
Grid\Get Pattern.vi
Grid\Get Position for XY Point.vi
Grid\Get XY Point.vi
Grid\Global To Graph Coordinates.vi

```
Grid\Global To Local Coordinates.vi
Grid\Global To Local Coordinates2.vi
Grid\Graph To Global Coordinates.vi
Grid\Grid.lvclass
Grid\Local To Global Coordinates.vi
Grid\Read Config.vi
Grid\Set Grid.vi
Grid\Set Pattern.vi
Grid\Write Config.vi

Icons\1393304015_519571-067_Pause.png
Icons\1393304096_icon-stop.png
Icons\1393304128_519570-066_Play.png
Icons\1393306789_519548-044_Repeat.png
Icons\1393307102_519548-044_Repeat.svg
Icons\1393307139_519548-044_Repeat.png
Icons\Add.png
Icons\Add.svg
Icons\crosshairs - 1x.svg
Icons\crosshairs - box.png
Icons\crosshairs - Box.svg
Icons\crosshairs -1x.png
Icons\crosshairs.png
Icons\crosshairs.svg
Icons\delete all.png
Icons\delete all.svg
Icons\delete.png
Icons\Delete.svg
Icons\line_select.png
Icons\line_select.svg
Icons\pan hand - Copy.png
Icons\pan hand - Copy.svg
Icons\pan hand.png
Icons\pan hand.svg
Icons\square.png
Icons\square.svg
Icons\square_select.png
Icons\square_select.svg
Icons\zoom in.png
Icons\zoom in.svg
Icons\zoom out.png
Icons\zoom out.svg

Layout\Get Grid.vi
Layout\Get Max Number of Points.vi
Layout\Get Max XY.vi
Layout\Get Name.vi
```

```
Layout\Get Position for XY Point.vi
Layout\Get XY Point.vi
Layout\Layout.lvclass
Layout\Read Config.vi
Layout\Write Config.vi

licences\RC4-0635_Pratt_BP4-1_Femto_HVA0240_PI4_LD4.lic
licences\RC4-0635_Pratt_BP4-1_Femto_HVA0240_PMD4_PI4_LD4.lic
licences\RC4-0682_Davis_BP4-1_Femto_HVA0240_PMD4a_LD4_PI4.lic
licences\RC4-0682_Davis_BP4-1_Femto_HVA0240_PMD4a_LD4_PI4-TTL.lic

Line\Get Grid.vi
Line\Get Line.vi
Line\Get Max Number of Points.vi
Line\Get Max XY.vi
Line\Get Name.vi
Line\Get Position for XY Point.vi
Line\Get XY Point.vi
Line\Line.lvclass
Line\Read Config.vi
Line\Set Line.vi
Line\Write Config.vi

Lockin\Calculate dI-dV.vi
Lockin\Config.vi
Lockin\Get Channels To Record.vi
Lockin\Get Config.vi
Lockin\Get Full Name.vi
Lockin\Get Name.vi
Lockin\Get Ripple.vi
Lockin\Lockin.lvclass
Lockin\Read Config.vi
Lockin\Set Channels.vi
Lockin\Set Ripple.vi
Lockin\Write Config.vi
Lockin Ref\Checkin.vi
Lockin Ref\Checkout.vi
Lockin Ref\Create.vi
Lockin Ref\Get Channels To Record.vi
Lockin Ref\Get Config.vi
Lockin Ref\Get Full Name.vi
Lockin Ref\Get Name.vi
Lockin Ref\Get Ripple.vi
Lockin Ref\Lockin Ref.lvclass
Lockin Ref\Read Config.vi
Lockin Ref\Set Ripple.vi
Lockin Ref\Write Config.vi
```

```
Points\Get Grid.vi
Points\Get Max Number of Points.vi
Points\Get Max XY.vi
Points\Get Name.vi
Points\Get Position for XY Point.vi
Points\Get XY Point.vi
Points\Points.lvclass
Points\Read Config.vi
Points\Write Config.vi

Scan\Add Point.vi
Scan\Delete All Points.vi
Scan\Delete All Scan Modules.vi
Scan\Delete Point.vi
Scan\Get Current Index.vi
Scan\Get Max Points.vi
Scan\Get Number of Points for Active Scan.vi
Scan\Get Number of Points.vi
Scan\Get Point.vi
Scan\Get Save Location.vi
Scan\Get Scan Layout.vi
Scan\Get Scan Speed.vi
Scan\Get Staus.vi
Scan\Read Config.vi
Scan\Scan.lvclass
Scan\Set Current Index.vi
Scan\Set Save Location.vi
Scan\Set Scan Layout.vi
Scan\Set Scan Speed.vi
Scan\Set Status.vi
Scan\Write Config.vi
Scan\Write Current Index.vi

Scan Module\Config.vi
Scan Module\Create.vi
Scan Module\Display.vi
Scan Module\Get Config Vi.vi
Scan Module\Get Display VI.vi
Scan Module\Get Name.vi
Scan Module\Get Number of Points.vi
Scan Module\Get Ready.vi
Scan Module\Grid to Points.vi
Scan Module\Line to Points.vi
Scan Module\Read Config.vi
Scan Module\Run.vi
Scan Module\Save Data.vi
```

```
Scan Module\Save.vi
Scan Module\Scan Module.lvclass
Scan Module\Update.vi
Scan Module\Write Config.vi
Scan Module Ref\Checkin.vi
Scan Module Ref\Checkout.vi
Scan Module Ref\Create From Name.vi
Scan Module Ref\Create.vi
Scan Module Ref\Get Config Vi.vi
Scan Module Ref\Get Display Vi.vi
Scan Module Ref\Get Namevi.vi
Scan Module Ref\Get Ready.vi
Scan Module Ref\Is Empty.vi
Scan Module Ref\Read Config.vi
Scan Module Ref\Run.vi
Scan Module Ref\Save Data.vi
Scan Module Ref\Scan Module Ref.lvclass
Scan Module Ref\Write Config.vi

Scan Ref\Add Point.vi
Scan Ref\Add Scan Module.vi
Scan Ref\Checkout Scan.vi
Scan Ref\Create Scan Ref.vi
Scan Ref\Delete All Points.vi
Scan Ref\Delete All Scan Modules.vi
Scan Ref\Delete Point.vi
Scan Ref\Delete Scan Module.vi
Scan Ref\Get Current Index.vi
Scan Ref\Get Grid.vi
Scan Ref\Get Line.vi
Scan Ref\Get Max Points.vi
Scan Ref\Get Number of Points.vi
Scan Ref\Get Number of Scan Modules.vi
Scan Ref\Get Point.vi
Scan Ref\Get Save Location.vi
Scan Ref\Get Scan Layout.vi
Scan Ref\Get Scan Module.vi
Scan Ref\Get Scan Speed.vi
Scan Ref\Get Status.vi
Scan Ref\Read Config.vi
Scan Ref\Scan Ref.lvclass
Scan Ref\Set Current Index.vi
Scan Ref\Set Grid.vi
Scan Ref\Set Line.vi
Scan Ref\Set Save Location.vi
Scan Ref\Set Scan Layout.vi
```

```
Scan Ref\Set Scan Speed.vi
Scan Ref\Set Status.vi
Scan Ref\Tip Withdraw.vi
Scan Ref\Write Config.vi
Scan Ref\Write Current Index.vi

Stanford Reseach\Calculate dI-dV.vi
Stanford Reseach\Get Channels To Record.vi
Stanford Reseach\Get Config.vi
Stanford Reseach\Get Full Name.vi
Stanford Reseach\Get Name.vi
Stanford Reseach\Read Config.vi
Stanford Reseach\Set Ripple.vi
Stanford Reseach\Stanford Research Config.vi
Stanford Reseach\Stanford Research.lvclass
Stanford Reseach\Write Config.vi

Tests\Approach Plot.py
Tests\Bias Spectroscopy Scan Run Wrapper.vi
Tests\Bias Spectroscopy.csv
Tests\Bias Spectroscopy.ini
Tests\Calculate PID for Speed.vi
Tests\createFit.m
Tests\Linear Space.vi
Tests\Log Space.vi
Tests\PID Graph 2D.py
Tests\PID Graph 3D.py
Tests\PID Graph.py
Tests\PID Speed - Individual.vi
Tests\PID Speed - loop.vi
Tests\PID_Graph_3D.m
Tests\read_from_file.m
Tests\Simple Run.vi
Tests\temp.py
Tests\Test Bias Spectroscopy Scan.vi
Tests\Test Bias Spectroscopy.py
Tests\test reaction speed 2.vi
Tests\test reaction speed.vi
Tests\Test Tip Extend.vi
Tests\Test Tip Walk.vi
Tests\Test Voltage Set2.vi
Tests\Tip Extend Feedback.vi
Tests\Tip Extend2.vi
Tests\Velocity vs PID.svg
Tests\Walk Plot.py

Topography\Get Config Vi.vi
```

```
Topography\Get Display VI.vi
Topography\Get Name.vi
Topography\Get Repeat.vi
Topography\Get Topography.vi
Topography\Read Config.vi
Topography\Run Nanonis Topography.vi
Topography\Run.vi
Topography\Save Data.vi
Topography\Save.vi
Topography\Set Repeat.vi
Topography\Set Save Location.vi
Topography\Set Topography.vi
Topography\Topography Config.vi
Topography\Topography Display.vi
Topography\Topography.lvclass
Topography\Write Config.vi
```